

PCTWORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

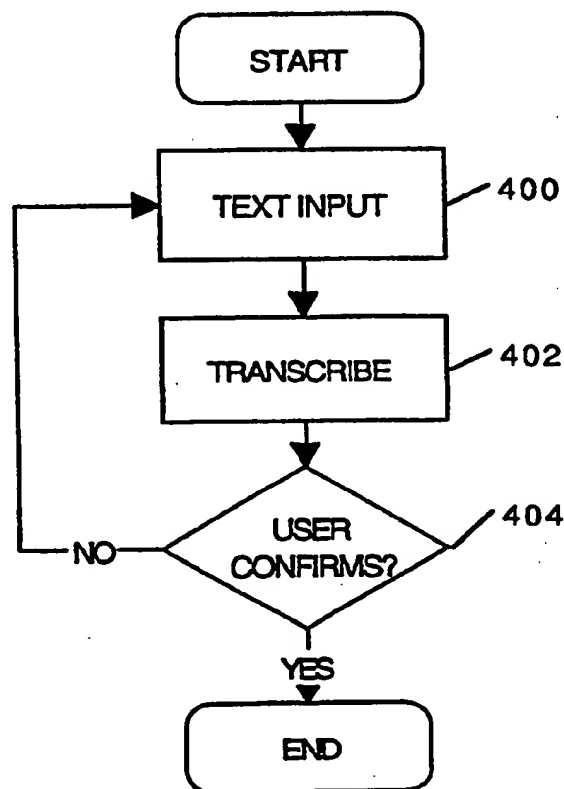
INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁶ : G06F 17/28		A1	(11) International Publication Number: WO 95/17729
			(43) International Publication Date: 29 June 1995 (29.06.95)
(21) International Application Number: PCT/US94/05586			(81) Designated States: AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, ES, FI, GB, HU, JP, KP, KR, KZ, LK, LU, LV, MG, MN, MW, NL, NO, NZ, PL, PT, RO, RU, SD, SE, SK, UA, UZ, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).
(22) International Filing Date: 18 May 1994 (18.05.94)			
(30) Priority Data: 08/171,591 22 December 1993 (22.12.93) US			
(71) Applicant: TALIGENT, INC. [US/US]; 10201 N. De Anza Boulevard, Cupertino, CA 95014 (US).			
(72) Inventors: COLLINS, Leland, D.; 251 Embarcadero Place, Palo Alto, CA 94301 (US). LIN, Judy; 1485 DeRose Way, San Jose, CA 95126 (US).			
(74) Agent: STEPHENS, L., Keith; Taligent, Inc., 10201 N. De Anza Boulevard, Cupertino, CA 95014 (US).			Published With international search report.

(54) Title: INPUT METHODS FRAMEWORK

(57) Abstract

A method and system for assisting input of information. The method and system dynamically translates information being input, and allows user interaction with the translation process. The system is flexibly designed to allow easy use by application developers and users. The design allows a common input method to be used by multiple applications. Users may also customize the input methods to suit their own needs. The input methods support "active" areas of text input, character based input, and phrase based input. The active area can be customized to appear a certain way. Dictionaries are also supported, including optional properties such as grammar and frequency. Dictionaries may also be chained.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	GB	United Kingdom	MR	Mauritania
AU	Australia	GE	Georgia	MW	Malawi
BB	Barbados	GN	Guinea	NE	Niger
BE	Belgium	GR	Greece	NL	Netherlands
BF	Burkina Faso	HU	Hungary	NO	Norway
BG	Bulgaria	IE	Ireland	NZ	New Zealand
BJ	Benin	IT	Italy	PL	Poland
BR	Brazil	JP	Japan	PT	Portugal
BY	Belarus	KE	Kenya	RO	Romania
CA	Canada	KG	Kyrgyzstan	RU	Russian Federation
CF	Central African Republic	KP	Democratic People's Republic of Korea	SD	Sudan
CG	Congo	KR	Republic of Korea	SE	Sweden
CH	Switzerland	KZ	Kazakhstan	SI	Slovenia
CI	Côte d'Ivoire	LI	Liechtenstein	SK	Slovakia
CM	Cameroon	LK	Sri Lanka	SN	Senegal
CN	China	LU	Luxembourg	TD	Chad
CS	Czechoslovakia	LV	Latvia	TG	Togo
CZ	Czech Republic	MC	Monaco	TJ	Tajikistan
DE	Germany	MD	Republic of Moldova	TT	Trinidad and Tobago
DK	Denmark	MG	Madagascar	UA	Ukraine
ES	Spain	ML	Mali	US	United States of America
FI	Finland	MN	Mongolia	UZ	Uzbekistan
FR	France			VN	Viet Nam
GA	Gabon				

INPUT METHODS FRAMEWORK

COPYRIGHT NOTIFICATION

5 Portions of this patent application contain materials that are subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

Field of the Invention

10 This invention generally relates to improvements in computer systems, and more particularly to a system and method for processing text input.

Background of the Invention

15 Technology has broken down many of the distance barriers previously encountered by business and society in general. Global communication has provided unprecedented immediacy of information in most forms. This has provided a corresponding level of exposure of cultures and events. While much of the communication which takes place is in the universal language of
20 images, there remains a large segment of communication and correspondence which requires various levels of translation.

Computer technology has made possible electronic processing of information in a variety of languages, which includes electronically assisting in the translation of language. Many devices have been devised to assist in
25 entering international information into documents, and creating documents which are not in the native language of the user creating the document. These devices are rigidly designed utilizing a self-contained unit of software. In other words, the use of the translating equipment is limited to the software itself. This inflexibility leads to a somewhat limited use of the equipment, and
30 correspondingly limits the supported environments.

In environments where a user inputs information, there is a need for a more flexible structure to overcome the limitations discussed above with respect to language processing.

Summary of the Invention

35 Accordingly, it is a primary object of the present invention to provide a system and method for overcoming the problems mentioned above with respect to translating and assisting in translating information.

-2-

It is a further object of the present invention to provide a flexible system and method for assisting a user in developing information in a form which the user is not familiar with.

It is still another object of the present invention to provide a system and method which allows multiple applications to incorporate a common
5 standardized input method in a flexible manner.

It is another object of the present invention to provide a system and method for providing a single extensible object oriented input method.

These and other objects are realized by an object oriented input method framework which provides a variety of useful tools to assist a user in the input
10 of information. The system includes objects for receiving user input, translating the input, allowing the user to modify the translation, and confirm that a translation is acceptable.

The input methods of the present invention can advantageously be at the
15 system level, which allows easy incorporation by application developers.

The input methods of the present invention can also be selected and modified by the user in order that each user customizes aspects of the input method framework to their own needs. The present system allows users to buy
20 off the shelf components of input methods to further customize their input method system. A preferred embodiment supports advanced features such as multiple active input areas, integrated undo, attaching phonetic information to text, and managing multiple input methods.

These and other objects and advantages will become evident from the specification and claims which follow.

25

Brief Description Of The Drawings

Figure 1 illustrates a typical hardware configuration of a computer in accordance with the subject invention;

Figure 2 illustrates an example of an ideographic sentence;

30 Figure 3 shows a typing sequence and the display corresponding to the typing sequence;

Figure 4 is a flowchart demonstrating the overall process of text input ;

Figure 5 shows what Japanese inline input might look while in progress;

Figure 6 demonstrates the processing of text using various input
35 methods;

Figures 7a, 7b and 7c illustrate various conversion examples ;

Figure 8 is a Booch diagram showing the text input method classes;

Figure 9 is a Booch diagram illustrating the conversion classes;
Figure 10 is a Booch diagram illustrating the dictionary classes;
Figure 11 is a Booch diagram of the active area classes;
Figure 12 illustrates alternate input sources;
5 Figure 13 illustrates processing an input method request; and
Figure 14 is a Booch diagram showing processing of new text and
conversion.

Detailed Description Of The Invention

10 The detailed embodiments of the present invention are disclosed herein.
It should be understood, however, that the disclosed embodiments are merely
exemplary of the invention, which may be embodied in various forms.
Therefore, the details disclosed herein are not to be interpreted as limiting, but
merely as the basis for the claims and as a basis for teaching one skilled in the
15 art how to make and/or use the invention.

Parts of the discussion below give examples which are specific to a certain
translation. As outlined above, however, the principles disclosed herein are
applicable in virtually any environment in which a user is inputting
information. The present invention is transparent with respect to the types of
20 information being handled. One goal of the present system is to provide a
flexible input system for virtually any data type.

The history of object-oriented programming and the developments of
frameworks is well-established in the literature. C++ and Smalltalk have been
well-documented and will not be detailed here. Similarly, characteristics of
25 objects, such as encapsulation, polymorphism and inheritance have been
discussed at length in the literature and patents. For an excellent survey of
object oriented systems, the reader is referred to "Object Oriented Design With
Applications" by Grady Booch.

While many object oriented systems are designed to operate on top of a
30 basic operating system performing rudimentary input and output, the present
system is used to provide system level support for particular features. It should
be kept in mind, however, that innovative objects disclosed herein may also
appear in layers above the system level in order to provide object support at
different levels of the processing hierarchy.

35 The invention is preferably practiced in the context of an operating
system resident on a personal computer such as the IBM ® PS/2 ® or Apple ®
Macintosh ® computer. A representative hardware environment is depicted
in Figure 1, which illustrates a typical hardware configuration of a computer in

accordance with the subject invention having a central processing unit 10, such as a conventional microprocessor, and a number of other units interconnected via a system bus 12. The computer shown in Figure 1 includes a Read Only Memory (ROM) 16, a Random Access Memory (RAM) 14, an I/O adapter 18 for
5 connecting peripheral devices such as disk units 20 and other I/O peripherals represented by 21 to the system bus 12, a user interface adapter 22 for connecting a keyboard 24, a mouse 32, a speaker 28, a microphone 26, and/or other user interface devices such as a touch screen device (not shown) to the bus 12, a communication adapter 34 for connecting the workstation to a data processing
10 network represented by 23. A display adapter 36 for connecting the bus to a display device 38. The workstation has resident thereon an operating system such as the Apple System/7 ® operating system.

An *input method* provides a mechanism for converting keyboard characters to ideograms: generally the Chinese characters used East Asian
15 writing. Since the repertoire for such a character set numbers in the thousands, it is not possible to type each character directly. Instead, the keyboard generates a small number of *components*, e.g., radicals or phonetic symbols, and an input method converts them into the appropriate ideograms. Examples of phonetic symbols include *kana* for Japanese, *hangul* for Korean, and *bopomofo* or the
20 English alphabet for Chinese.

The preferred embodiment disclosed herein describes external interfaces of what will be referred to as the "Input Method Framework." The following discussion offers some background information for those unfamiliar with ideographic input, summarizes the features, identifies the goals that drive the
25 design, provides detailed description of the framework classes, and uses object diagrams to describe some common scenarios.

Imagine for a moment that English were written with logographs. Each separate word or syllable was represented by a separate picture, and is written without spaces. The number of words is clearly too large to represent on a
30 keyboard, and additional processing is needed to input characters.

The most popular method allows the user to type in characters in a phonetic transcription, which is automatically translated into ideograms (other techniques—including handwriting analysis in conjunction with pen input—are also possible, and can be used in addition). Let's suppose that we wanted to
35 enter an ideographic sentence having the artificial logographs listed in the Fake Logograph column of FIG. 2.

The artificial logographs in this example are the in the table of FIG. 2. We also include real Chinese logographs for comparison.

FIG. 3 shows the typing in sequence. We would type our desired sentence in phonetically as: *unayburredhurudres* (remember that there are no spaces!). As we type, the input method parses the text grammatically, and converts phrases to the corresponding ideograms. Since there may be a number of renderings, the text changes as we type.

Even once the sentence is complete, the ideograms are not correct. It requires additional knowledge beyond syntax to get a correct result, since syntactically both the forms: Det·Noun·Verb·In. Object· Det·Noun, e.g. "A neighbor read her a book", and Det·Noun·Verb·Pos. Pronoun· Noun, e.g. "A neighbor read her address" are syntactically correct. So, the user needs to modify the transcription by choosing alternative renderings.

This task is done by either modifying the phrase length (in this case, indicating that *udres* is one word), or by selecting an alternative ideogram (e.g. manually choosing the book ideogram instead of the apple ideogram).

Unfortunately, most sentences are much more complex than this example—especially in Japanese—and have many more alternative phrase lengths and alternative readings. The number of homophones (ideograms with the same pronunciation) in Japanese and Chinese are very large: up to one hundred readings for the same symbol! Even when alternatives are chosen, the input method uses its knowledge of the grammatical structure of the sentence to filter out inappropriate options; otherwise the user would be swamped. (The user can choose to see all options for unusual readings.)

Inline Input: Japanese Example

FIG. 4 is a flowchart demonstrating the overall process of text input. Text editing classes can use an input method to provide *inline input*, where the text is entered and converted directly in the document (as opposed to in a separate popup window). As the user types, text is displayed in the *active area*. The newly entered text should be visually distinct from the rest of the document so that the user can see which text is being processed. As typing continues, the contents of the active area changes. For example, romaji input is transcribed into kana by transliterators, and new phrases are identified and converted by the input method. When the user is satisfied with the conversion results, he *confirms* the text and collapses the active area.

At any given time, there are a number of different kinds of text on the screen:

- *Confirmed Text*: Text that has already be confirmed and is no longer processed by the input method.

- *Converted Text*: Text for which the input method has returned a result. This text is still within the active area and can be adjusted through the input method.

- *Raw Text*: Unconverted text in the active area, generally phonetic characters.

5 FIG. 5 shows what Japanese inline input might look while in progress. The active input area 500 is outlined with a solid box. Notice that it starts in the middle of the first line and ends in the middle of the second line. The text outside of the active area does not participate in the conversion process. The dotted boxes denote phrases, or segments 502, which have already been
10 converted. There are also un-converted, but transcribed, *kana* characters 504, 506, which are followed by a single romaji letter 508. The insertion point is after the romaji letter.

Input Method Operations

15 This section describes some other operation a user may need to use when typing Japanese sentences. FIG. 6 demonstrates the processing of text using various input methods. The general process involves input of text 600, conversion of the text, which is shown to the user at 602, user interaction with the text at 604, and either converting the text with the user selections at 606, and
20 confirming the conversion process 608.

Select phrase. First, the user must indicate which phrase is to be adjusted. This could be done by pointing at the phrase with the mouse, or by typing some sequence of commands, such as the left and right arrow keys. However the phrase is selected by the user, it must be given a visual appearance different
25 from the rest of the document text, and the rest of the text in the active area.

Change phrase boundary. The user can change the segmentation by indicating that a phrase should be made longer or shorter. (Note that this makes the following phrase shorter or longer, and is thus the same as moving the phrase boundary.) This could be done by dragging the phrase boundary with the
30 mouse, or by typing some sequence of commands, such as the up and down arrow keys. Depending upon the particular input method, changing the length of a phrase may cause all the phrases following it to change as well.

Choose homophone. The user can display alternate homophones for a phrase by double-clicking on the phrase, or by typing some sequence of
35 commands, like the enter key. Alternatives can be selected either by changing to the next homophone in-place, or by bringing up a floating window displaying all homophones, from which the user would then choose the correct homophone.

Confirm conversion. The user can indicate that conversion is complete by some command like return or enter. When this is done the distinctive visual appearance of the active area would be removed and the caret would be placed after the newly converted text.

5 Different levels of functionality could be supported by different input methods. For example if the user were to delete a character from the middle of a phrase, the information that an input method maintains to process the active area could become invalid. Thus some input methods may disallow such deletions. Most, however, allow the user to freely edit the un-converted input.
10 Less sophisticated input methods might wait until the whole sentence is typed before doing any segmentation, while other input methods might generate phrases "on the fly". Some input methods might decide to "drop" phrases off the front of the active area as the user types in order to limit the amount of internal storage needed.

15

Input Method Dictionaries.

In order to support the conversion process described above, input methods make use of large dictionaries. These dictionaries typically provide the ability to make queries using a component *key* to lookup a result *entry*. For
20 example, as shown in FIG. 7(a), in parsing the *kana* input for the sentence, "In Spring, Nagasaki" a Japanese input method engine might search one or more dictionaries using the component *kana* to determine which substrings of the input text are possible words. In many cases, one key will map to multiple entries (typically Chinese characters plus trailing *kana* results). For example, as
25 shown in FIG. 7(b), the first two *kana* (*ha, ru*) map to entries 1, 2, and 3.

In the next phase, the input method might use grammatical information stored with each entry to find the most appropriate result for the given context. In the example above, the first entry, "Spring" is matched because it is a noun that can take the locative particle that follows it. The others are unlikely
30 matches because of their grammatical categories.

Frequency information might also be used to weed out less likely options or to order the options for presentation to the user when the user wants to pick an option other than that chosen by the input method.

A single result might also be arrived at via different components, as
35 shown in FIG. 7(c).

There are two primary categories of input method dictionaries: the main dictionary plus one or more user dictionaries. The dictionaries are large enough to account for typical language usage, but are seldom user-editable. The

user dictionaries are designed specifically to be user editable so that the user can provide the input method with hints for selecting result characters that are better suited to the user's particular usage. Another type of user dictionary is a dictionary designed for use in specialized fields such as medicine, law, etc.

5 These are generally treated the same as user dictionaries by the input method, but the developer might disallow user editing. Most input methods allow the user to select the dictionaries which will be searched and to determine the search order, with priority given to entries in the user dictionary over entries in the common dictionary.

10 Currently, dictionaries designed to work with one input method rarely work with any other input method. This prevents users from using their user dictionaries with different input methods. The primary clients of the Input Method Dictionary classes are developers of input methods. Other potential clients are developers of dictionaries for specialized fields such as law and
15 medicine.

Goals and Features

Entering text with a keyboard is likely the least exciting yet most basic aspect of using a computer. English-speaking users take for granted the ability
20 to type with a keyboard in exactly the same way, regardless of system or application context. Yet for many Chinese and Japanese users, the inability to input text in an efficient, consistent and intuitive manner stands as the biggest hurdle to effective computing.

The Input Method Framework's primary purpose is to address this
25 problem by supporting *system-level* integration of input methods with other subsystems such as the Document Architecture, and the Typing Framework. It also provides base-level functionality including inline conversion, and the saving and restoring of component information. The framework supplies these services through a set of extensible objects. By adhering to the interface, an
30 input method developer ensures that his product can be used for text editing throughout the system.

The following sections elaborates on primary and secondary design goals.

Primary Goals

35 The following points are fundamental ideas that drive the design.

- *Allow the user to buy off-the-shelf input method components and use them to enter text everywhere in the system. Imagine using one keyboard to type with*

the Finder, another for MacWrite, and yet another with Microsoft Word! This is currently the situation faced by Asian users.

- *Provide support for selecting and synchronizing input method.* This control process can be very complex. For Japanese alone, there are *five* modes of input:

5 kanji conversion, katakana, hiragana, half-width katakana and half-width hiragana. The situation is further complicated if the user uses other scripts like English. It will allow different input methods to be freely mixed without losing context information. For example, the user may wish to use a radical-based input method for a single character in the midst of *kana->Kanji* conversion.
10 Switching to the radical-based method should preserve the phonetic method's context.

- *Provide a common dictionary access protocol to enable data sharing.*

Minimally, we must allow user-defined dictionaries to be shared by different conversion products. A common format would also enable special purpose
15 dictionaries such as medical or legal dictionaries to be sold and marketed separately. There are numerous possible strategies which could be used to implement the dictionaries, including hardware-independent dictionaries. A different dictionary could be developed for every hardware target.

- *Provide full integration with Local Text and the Document Architecture.*

20 Integration with the local text system enables inline input throughout the system. Any developer who base his product on the local text automatically receives input method functionality.

- *Provide flexible undo support.* Most current products do not support undo because 1) input methods are highly context sensitive, and it is difficult to undo
25 an operation since it may be impossible to return to a previous state, and 2) it has not been clear where the responsibility lies between the application and the input method. We provide objects to facilitate undo implementation. The input method developer can use these objects to store information needed to retrieve previous context, and to subclass to customize undo behavior. One
30 difficult problem is determining what constitutes an undo-able unit of typing when an input method is involved. That is, it must be determined with respect to each conversion result whether the conversion result could be considered separately undo-able. Requests such as selecting an option may or may not be considered undo-able. The entire typing sequence could be considered undo-
35 able

- *Supply human interface guidelines and examples for input method developers. Implement user interface elements with default behavior to promote consistency.* The preferred embodiment will provide input method

implementations with the system. These efforts can serve as examples for third party developers. Supplying a set of user interface elements that are common to all keyboard input methods, e.g., option window, status view, etc., is also contemplated. While the general goal is to provide a common interface, it is also possible to give the developer the choice of using the common interface in order that the developer not be deprived of the flexibility of using their own interface.

- *Achieve acceptable typing performance.* Typing speed should not degrade noticeably when using an input method. Minimally, the speed should be sufficient for a data entry professional.

- *Allow existing conversion products to be easily ported without sacrificing architectural integrity.* Input method developers are usually multi-platform suppliers, and their products are designed to be portable. Since users grow attached to their input methods, we want to accommodate existing products as long as we do not compromise the architecture.

- *Design the framework such that it is flexible enough to accommodate non-keyboard text entry such pen input or visual (i.e., point-and-click) radical-based input.* The developer should not have to circumvent the framework to support other devices.

- *Interchangeable dictionaries:* The dictionaries used for input methods should be usable by different input methods. Although a dictionary may be designed and optimized for a particular input method, the common protocol of the Input Method Dictionary classes will allow access by any input method. This feature is particularly desirable for user and specialized dictionaries, but should be usable with all input method dictionaries. Developers of specialized dictionaries should be able to implement their products knowing that they can be used by any input method.

- *Dictionary editing:* The Input Method Dictionary classes will provide a protocol to support user and programmatic editing (adding, modification, and deletion of entries) of input method dictionaries.

- *Flexible dictionary search criterion:* The entry lookup methods should permit flexible search criteria, including possibly the use of regular expressions. For example, clients should be able to lookup entries in a Chinese dictionary with or without tones:

beijing => 1. bei3jing1 (北京) 2. bei4jing3 (背景)

- *Dictionary size:* Because input method dictionaries contain a large amount of data, compression is often been a critical issue in how they are structured.

While this is not an overriding concern, we will need to pay some attention to this issue.

2. Secondary Goals

- 5 • *Provide an extensible architecture that can be used by other linguistic services.* The framework should accommodate other "plug-in" components such as spell checker, grammatical analysis tools or hyphenation engines. These services are usually batch oriented and can be implemented with commands in a straight forward manner. However, the input method architecture allows these
10 operation to be invoked at text entry time.
- *Allow all text editing classes to use the input methods.* Some developers may wish to write their own text editing classes. The framework should allow these clients transparent access to input method objects.
- *Support multiple active areas.* Most products today only keep the context
15 information for a single active area. If the user sets a different insertion point (even if it is in a different document), his previous input is confirmed and the states associated with that conversion area is lost. We want to support an arbitrary number of active areas over all documents. The framework is responsible for activating and deactivating the appropriate active area, and for
20 coordinating between active areas and encapsulators.
- *Provide the ability to store and retrieve component information; maintain the consistency of this information during text editing.* This support allows us to return confirmed text to an active state and be re-edited. It also provides the information necessary to support *furigana*, where phonetic information is
25 displayed along side *kanji* characters. There are many possible ways to support *furigana*. One alternative is to store the component information as style runs. This has the advantage that keeping the phonetic in sync during editing is handled by the style mechanism. The disadvantage is that having so many style runs causes text display to slow down significantly.
- *Provide a generic input method human interface implementation that can be used with multiple conversion engines.* It seems advantageous to have a single
30 interface for entering all Chinese, Japanese and Korean text. However, current implementations all have a very tightly coupled relationship between the conversion engine and the input method interface; so this goal may be difficult
35 for existing products. In any event, this generic input method will serve as our example for human interface design.

-12-

- *Provide a set of styles that developers can use to denote text in different conversion states.* Which styles to use will also be included in the human interface guidelines.

5 3. Other Goals

- *User interface for selecting input methods.* While this design provides the API for selecting and synchronizing multiple input methods, another Framework will be relied on to determine the user interface. The present embodiment could be designed to include this feature.
- 10 • *Implement conversion engines.* Conversion engines require a great deal of expertise in language and grammar syntax. This task is probably best left to third party developers with such knowledge. Several conversion methods could be included with the system, and they could be ported from an existing implementation.
- 15 • *Improve conversion accuracy.*
 - *Supply input method dictionaries.* Again, this task is probably a third party opportunity. The present embodiment does contemplate a Chinese phrase-level dictionary. A good dictionary makes Chinese conversion relatively straight forward.
- 20 • *Installation of input method dictionaries:* This functionality will be provided primarily through the file system and the installation framework, although the present embodiment could be designed to include such a feature. How installation of input method dictionaries is done will be transparent to clients of the Input Method Dictionary classes.
- 25 • *Provide user interface editors and views.* There are a number of tools needed to edit and view components like the command key mapping, user dictionary, etc.

Class and Method Description

30 This sections describe classes in the Input Method Framework. After an overview, it presents class diagrams for each category as well as description of each class. For the more interesting classes, the public and protected methods are also discussed.

35 Architecture Overview FIG. 8

The objects and relations among objects shown and discussed below are merely examples of the overall concept of the input method framework.

The clients for this framework are primarily input method developers, and a Typing Framework. Text editing classes may be interested in TTextModifier 804. Access to TTextModifier 804, however, should be through the TTypingConfiguration object 802. Access to TTypingConfiguration 802 is
5 through TTextEncapsulator 800. The classes in this framework can be divided into the following categories:

Text Input Method Classes: These classes implement the input method user interface. They work in conjunction with Text Frameworks to handle user
10 events and to modify the text object.

- TTextModifier 804
 - TTextInputMethod 806
 - TCharacterBasedInputMethod 814
 - TPhraseBasedInputMethod 820
 - TTransliterator
- TTextInputMethodCommand
 - TExplicitConversionCommand
 - TUnconvertCommand
 - TSelectNextPhraseCommand
 - TSelectPreviousPhraseCommand
 - TBreakApartPhraseCommand
 - TMergePhraseCommand
 - TConfirmCommand
- TInputMethodAppearance 808
 - TCharacterBasedAppearance 810
 - TPhraseBasedAppearance 812

Conversion Engine Classes: These classes define and implement conversion engines.

- TInputMethodConversionEngine
 - TChineseConversionEngine
 - TKoreanConversionEngine
 - TSJIPConversionEngine

Dictionary Classes: These classes define and implement input method dictionaries.

- TInputMethodDictionary
 - TInputMethodDictionarySurrogate

- TInputMethodDictionaryIterator
- TInputMethodDictionaryKey
- TInputMethodDictionaryEntry
- TInputMethodDictionaryEntryIterator

5

Active Area Classes: These classes store and manipulate component information in active areas.

- TInputMethodPhrase
- TInputMethodPhraseSequence
- TInputMethodActiveArea 818

10

User Interface Utility Classes: These classes implement some of the user interface elements for input methods. They can be used as is, or subclassed to customize behavior.

15

- TInputMethodEventMap
- TAlternateInputSource
 - TViewBasedAlternateInputSource
- TOptionSelector
 - TTrackerBasedSelector
 - TViewBasedSelector

20

Text Input Method Classes

TTextModifier 804

25

TTextModifier 804 is an abstract base class that defines the interface for filtering events from text application. All linguistic services that are invoked at the time of text entry should descend from this class.

Command Objects: TTextModifier 804 should update the model with command objects. Multiple commands can be grouped by

30

TModelCommandGroup to define a single undo-able unit. Similarly, if screen updates should not occur until several related commands have all finished executing, these commands should also be grouped into a single unit.

Interactive modifiers: A text modifier is interactive if it requires user intervention. For example, transliteration always has a deterministic result and is not an interactive modifier, where input methods or spell checkers typically require the user to select among a list of alternatives.

35

Chaining modifiers and relationship with the Typing Framework :
Modifiers can be grouped into an ordered list. This list can be used as a single

unit by the text editing client. Modifier lists are controlled by and accessed through, for example, a Typing Framework's TTypingConfiguration objects. We restrict each chain to contain only a single interactive modifier. In general, text editing classes should not use a modifier directly, but through the

5 TTypingConfiguration object 802. It is also possible that multiple interactive modifiers such as input methods could be allowed. It is also possible to provide direct use of the modifier by the user.

TTextInputMethod 806

10 TTextInputMethod 806 derives from TTextModifier 804 and defines the abstract protocol for all input method implementations. It is responsible for the user interface of input method, e.g., user input, option selection, or text appearance. It is not responsible for the actual conversion. This separation allows us to use the same UI for multiple conversion engines. This class is

15 responsible for services that are common to all input methods. These features include:

User Event Mapping: TTextInputMethod 806 uses a sub-object, TInputMethodEventMapping, to interpret events. This feature allows users to easily customize command mapping. Each event is mapped to a

20 TInputMethodCommand object that implements the requested action.

Appearance of the Active area: TTextInputMethod 806 uses a sub-object, TInputMethodAppearance 808, to define the visual appearance of the active area. The text classes have no knowledge of the different states during input. Styles (or attributes) are used to distinguish the active area. Each type of text

25 displayed, e.g., raw, converted or confirmed, has a distinct style.

Coordinating multiple active areas: The active area information is kept with the model so that it stays in sync. Each model can contain multiple active areas, where the current area corresponds to the current selection. This class is responsible for getting the current area for its subclasses.

30 *Coordinating multiple input methods:* The input method classes work in conjunction with the Typing Framework to coordinate multiple input methods. Each input method is responsible for updating the model when it is active, and for reconstructing its internal states from the model when reactivated. This support allows input methods to be freely mixed without

35 losing context information. A special case arises when the user switches typing configuration from one with an input method to one without. The second configuration is still responsible for updating the active area consistently by using a "null" input method.

TPhraseBasedInputMethod 820

TPhraseBasedInputMethod 820 is a concrete subclass of TCharacterBasedInputMethod 814. It implements the default user interface for phrase-based input methods. It implements functionality specific to a phrase based method such as changing phrase boundaries.

Input Method Command Classes

TInputMethodCommand

TInputMethodCommand descends from TModelCommand. It is an abstract base class for all input method commands. Each operation (ie., opening an option window) should be a concrete subclass of TInputMethodCommand.

Typically, a single TInputMethodCommand issues one or more other commands. These commands are put into a TModelCommandGroup and treated as a single undo unit. Developers may subclass to encapsulate other information it needs for undo by overriding the HandleUndo()/Redo() methods.

TInputMethodCommand objects are created by a TTextInputMethod 806. They use a TInputMethodAppearance object 808 to build the text inserted into the model.

There is a one-to-one correspondence between the commands and events initiated by the user such as down-shift key to open option window. The following concrete subclasses are used by TTextInputMethod:

TExplicitConversionCommand: Get the best-guess conversion from the engine and update the text model.

TUnconvertCommand: Given the current phrase, updates the model by replacing the converted text with raw text. Note that this feature is provided in addition to undo so the user can unconvert any arbitrary phrase, and does not need to traverse the command stack.

TSelectNextPhraseCommand: Highlights the next phrase, which becomes the current phrase.

TSelectPreviousPhraseCommand: Highlights the previous phrase, which becomes the current phrase.

TBreakApartPhraseCommand: Change the phrase boundary by breaking the current phrase into two.

TBreakAndConvertPhraseCommand: Change the phrase boundary by breaking the current phrase into two and attempt to convert. Updates the model with the new conversion.

TMergePhraseCommand: Change the phrase boundary by merging the current phrase with the one adjacent to it.

TMergeAndConvertPhraseCommand: Change the phrase boundary by merging the current phrase with the one adjacent to it and attempt to convert.

5 Updates the model with the new conversion.

TConfirmCommand: Removes active area styles from the text model.

TInputWindowCommand: Creates a separate input window. When the user indicates that he is finished with input, updates the model with text in the input window.

10 TSelectOptionCommand: Concrete base class for the command used to select homophone options. Given the current phrase, allows the user to select options directly in the document.

TSelectOptionInViewCommand: TSelectOptionInViewCommand is a concrete subclass of TSelectOptionCommand. It uses a TViewBasedSelector to select options in a separate view. Updates the model when the selection is confirmed.

4. Active Area Appearance Classes

20 TInputMethodAppearance 808

TInputMethodAppearance is an abstract base class that defines the protocol for specifying the active area appearance, something that is highly specific to each product. It is anticipated that most developers will provide their own subclass objects. This object contains a collection of TStyleSets (to become
25 TAttributeGroup), each corresponds to a phrase state (ie., converted, unconverted, etc.)

TCharacterBasedAppearance 810

This is a concrete subclass used in conjunction with
30 TCharacterBasedInputMethod. It implements a character based appearance, that is, the boundary between phrases is not delineated. If system-defined styles are not sufficient for this purpose, we may also supply some specialized input method styles. Specific styles could be supplied via inheritance or through data.

35 TPhraseBasedAppearance 812

This is a concrete subclass of TCharacterBasedAppearance 810, and is used in conjunction with TPhraseBasedInputMethod 820. It implements a phrase

based appearance, that is, the boundary between phrases is delineated through special styles.

Conversion Engine Classes FIG. 9

5 TTextInputConversionEngine 914

TTextInputConversionEngine 914 defines the protocol for converting data. It is the abstract base class for all input method conversion engines. Different engine objects can be used with the same TTextInputMethod 902. The engine uses the active area 900 passed into the Convert() method to help with the conversion. Note that while the conversion result could change portions of the active area if new input provided additional context, the engine never updates the area directly. Instead, TTextInputMethod 902 uses the conversion result in its command objects to update the model and the active area at the same time.

15 TTextInputDictionary 904 is discussed in greater detail with respect to FIG. 10.

TTextInputConversionEngineSurrogate is a concrete class that provides a light-weight mechanism for clients to access a TTextInputConversionEngine 914.

20 TSJIPConversionEngine 908: TSJIPConversionEngineTask is a wrapper object for the SJIP 3.0 Japanese input method product, our system default Japanese input method.

TChineseConversionEngine 912: TChineseConversionEngine is a wrapper object for our system default Chinese conversion engine.

25 TKoreanConversionEngine 910: TKoreanConversionEngine is a wrapper object for our system default Korean conversion engine.

Dictionary Classes

TInputMethodDictionaryKey 1006 is a concrete class used to lookup entries in an input method dictionary. Each key contains a component TText and optional properties (grammar, etc.). The properties allow the client to specify more precise conditions for queries. This is particularly useful in separating homophone options. The standard set of properties are the TTokens defined in TInputMethodDictionary 1002. Properties contained in a key need not be mutually exclusive. Properties are shared between the TInputMethodKey and TInputMethodEntryClasses.

The dictionary key will also allow the specification of other search criteria, including the regular expression specification.

Particular dictionaries may want to subclass from TInputMethodDictionaryKey 1006. For example, a subclass, TRadicalStrokeDictionaryKey 1008, adds the ability to get and set the components (radical and stroke) as separate numeric values.

5 TInputMethodDictionaryEntry 1004 is a concrete class used to represent an entry in the dictionary. Each entry contains a TText result and properties. While most dictionaries will not actually store their data in this format, the protocol in this class defines the least common denominator for what each entry should contain.

10 Particular dictionaries may want to subclass from TInputMethodDictionaryEntry 1004.

TInputMethodDictionaryEntryIterator is a concrete class used to traverse the entries in a dictionary. The primary clients will be a dictionary editor.

15 TInputMethodDictionary 1002

TInputMethodDictionary 1002 is an abstract class and defines the protocol for all input method dictionaries. It does not provide any implementation. This protocol includes methods for looking up, adding, and deleting entries. In addition, each dictionary contains a name (TLocalName), a category (main, user, specialized), name of the input method for which the dictionary is optimized, language (Japanese, Chinese, Korean, etc.) and possible an input component type. It serves as base class for all input method dictionaries, and should act as a wrapper object for all ported dictionaries.

20 *Properties:* Queries can be modified by properties. Properties (especially grammatical properties) can be used to determine the best fit among multiple entries.

Reverse mapping: This class includes protocol for reverse mapping from result to component form. Reverse mapping is an alternative way to retain component information without attaching any additional data to the document. The conversion engine can reconstruct the context it needs by looking up the component text phrase by phrase. Note that this ability comes at the expense of increasing the dictionary size. Furthermore, some results cannot map to a component form (although for the vast majority of cases this is not a problem).

35 *Data Sharing:* Since these dictionaries can be very big, the data needs to be shared by different teams. A framework for sharing will be developed as part of the dictionary design.

TInputMethodDictionarySurrogate 1012 is a concrete class that provides a light-weight mechanism for clients to access a TInputMethodDictionary 1002. Its semantic is explicit master with multiple surrogates.

5 TStandardTextInputDictionary 1010 is our default implementation of input method dictionary. It is a subclass of TInputMethodDictionary 1002 and has a TDiskDictionary. This hardware independent class will be optimized for input methods use. It is used for all user dictionaries, and developers of new input method dictionaries should use this class directly so they can benefit from future enhancements.

10

TInputMethodDictionaryGroup

Input method dictionaries may be chained. For example, one may wish to have a personal dictionary followed by a medical one, with the generic dictionary as last in the group. TInputMethodDictionaryGroup is responsible for
15 managing the chain, and allows clients to look up entries in all of the dictionaries in the group. This class is intended to be used directly and not as a subclass.

Active Area Classes FIG. 11

20

TTextInputConversionEngine 1104, TTextInputActiveArea 1102, and TTextInputMethod 1108 have been discussed previously with respect to FIG. 9. TTextEncapsulator 1100 has been discussed previously with respect to FIG. 8.

25 TTextInputPhrase 1106

TTextInputPhrase 1106 encapsulates the information about a phrase (aka clause or *bunsetsu*), mainly the component and result text, and the phrase state. It also contains information necessary to reconstruct the context (e.g., part of speech).

30

Phrase state: A phrase can be in one of several states defined by a TToken:

- *Converted:* The phrase has been converted, and the conversion has been incorporated into the document.
- *Manually converted:* The phrase was manually adjusted by the user, and the engine should not attempt any further conversions.
- 35 • *Unconverted:* The user issued an "undo" or chose the "Unconvert" operation and has returned the text into component form.
- *Deleted:* The phrase has been deleted by the user.

User Interface Utility ClassesTInputMethodEventMap

This class translates user events into input method operations. It maps a TEvent object to the TInputMethodCommand object that should be invoked.

5 Developers can add new actions by defining a new command objects, and register the event and command in the TInputMethodEventMap. Although most of the events in this object are TKeyEvents, the interface is generalized to all subclass of TEvent. This class is designed to be used directly and not as a base class; it is used internally by TTextInputMethod and its subclasses. In the future,

10 there will be an editor for this object.

Alternate Input Source ClassesTAlternateInputSource 1202

Input methods often takes input from sources other than the document.

15 For example, input can come from a "bottom-line input" window, a graphical radical-stroke composition utility, or from the homophone option selector. TAlternateInputSource 1202 is an abstract base class for these objects. It provides methods to update the target text model for its subclasses. The source of input can be either view-based (i.e., bottom-line input), or some other mechanism

20 (i.e., the tracking homophone selector). While a TAlternateInputSource 1202 object is active, it becomes the current target, and the text model will not receive any user events.

TViewBasedAlternateInputSource 1220 is an abstract base class for view-

25 based alternate input sources. It descends from TAlternateInputSource 1202 and TTextView 1218. Some possible subclasses of TViewBasedAlternateInputSource 1220 are bottom line input, graphical radical stroke input method, and homophone selection in a window.

TOptionSelector 1204 is an abstract base class for managing conversion

30 options (most typically homophones). This class defines the protocol for processing option, and it is up to its subclasses to supply the appearance. For example, options can be displayed in a separate window, in a pop-up menu, a list-view, or inline with the text.

TTrackerBasedSelector 1208 descends from TOptionSelector 1204,

35 TAlternateInputSource 1202, and TTracker 1206. It implements in-document option selection.

-22-

TViewBasedSelector 1214 descends from TOptionSelector and TViewBasedAlternateInputSource. It implements option selection in a separate window.

5 Examples and Tutorials

Using the Input Method Dictionary Classes

```

TInputMethodDictionaryIterator* dictionaryIterator =
    TInputMethodDictionary::CreateIterator();
10 TInputMethodDictionarySurrogate* dictionary =
    dictionaryIterator->First();
Boolean found;
TToken kJapanese("Japanese");
TToken dictLanguage;
15 while (dictionary && !found)
{
    dictionary->Language(dictLanguage);
    if (dictLanguage == kJapanese)
        found = TRUE;
20     else
        dict = dictionaryIterator->Next();
}
// ....Later
// Dictionary lookup
25 TText componentText(L"はる");    // When the compiler supports it
TInputMethodDictionaryKey key;
key.SetComponent(componentText);
TArray options;
TArray nouns;
30 dict.LookupOptions(key, options);
// options now contains entries with results:
//      1.春  2.張る 3.晴  4.貼る
key.AddProperty(TInputMethodDictionary::kNoun);
key.Filter(options, nouns);
35 // nouns contains
//      1.春
key.AddProperty(TInputMethodDictionary::kProperNoun);
key.Filter(options, nouns);

```


-23-

```

// nouns now contains
//      1.春  2.晴
TInputMethodDictionaryIterator entryIterator(dictionary);
TInputMethodDictionaryEntry* entry;
5 // Set the iterator to look at all the verb entries with
// the component "はる".
key.RemoveProperty(TInputMethodDictionary::kNoun);
key.RemoveProperty(TInputMethodDictionary::kProperNoun);
key.AddProperty(TInputMethodDictionary::kVerb);
10 entryIterator.SetIteratorAt(key);
while (entryIterator.IsValid())
{
    entry = entryIterator.CurrentAndIncrement();
    // do something to the entry
15 }

```

Scenarios

This section uses various figures to outline a few interesting scenarios. The description is a rough sketch of the sequence of events, and is not meant to be comprehensive. TTextInteractor 1300 detects events from the user. TTypingConfiguration 1302 determines what modifier group should be used. TTextInputMethod 1304 has been described above with respect to 806. TTextInputMethodRequestRegistry 1306 is a command created by the TTextInputMethod 1304. The TTextInputMethodRequestRegistry 1306 then creates the command object TTextInputMethodCommand 1310. TTextInputMethodCommand 1310 then updates the TTextEncapsulator 1308. The general flow of processing is discussed below.

1. Processing an Input Method Request Event - FIG. 13

Filtering events (steps 1, 2, 3): Text presentation receive user events and filters them through current chain of TTextModifiers by calling TTypingConfiguration::WantEvent(). This method determines the correct modifier group, and iterates through the group calling each modifier's WantEvent() method. It returns TRUE if any modifier's WantEvent() returns TRUE.

Process event (steps 4, 5): If TTypingConfiguration::WantEvent() returned TRUE, TTextInteractor calls ProcessEvent(). This method will again iterate over the group calling each individual ProcessEvent().

-24-

Executing the input method command (steps 6, 7, 8):

TTextInputMethod::ProcessEvent() uses TTextInputEventMapping to map the event to a particular request. It then creates the command object and updates the model.

5

2. Processing New Text and Conversion Results

Handle new text (steps 1, 2, 3): TTextPresentation 1400 buffers the keystrokes and passes the new text to the text modifiers through the typing configuration 1402. TTextInputMethod 1404 has been described above with respect to FIG. 8.

10

Component conversion (steps 4, 5, 6, 7): The input method 1404 calls the conversion engine 1406, which uses the active area 1408 for context information, and then looks up the new text in the dictionary 1410. If conversion is possible, return the result to the presentation which will update the encapsulator with a command.

15

While the invention has been described in terms of a preferred embodiment in a specific system environment, those skilled in the art recognize that the invention can be practiced, with modification, in other and different hardware and software environments within the spirit and scope of the appended claims.

20

CLAIMS

Having thus described our invention, what we claim as new, and desire to secure by Letters Patent is:

- 1 1. An apparatus for assisting in the input of information comprising:
 - 2 (a) processor means;
 - 3 (b) storage means attached to the processor means for storing information;
 - 4 (c) display means attached to the processor means and the display means for
5 displaying said information;
 - 6 (d) an input methods framework for creating and managing input method
7 objects including:
 - 8 (1) at least one object for receiving user input and outputting
9 converted user input in response to said user input; and
 - 10 (2) means for displaying the user input and the translated user input
11 on the display means.
- 1 2. The apparatus of claim 1, wherein the input methods framework
2 includes at least one object for processing a phrase in response to said
3 user input.
- 1 3. The apparatus of claim 2, wherein the at least one object for processing a
2 phrase includes means for processing a user-selected phrase.
- 1 4. The apparatus of claim 2, wherein the at least one object for processing a
2 phrase includes means for changing a phrase boundary of the phrase.
- 1 5. The apparatus of claim 1, wherein the input methods framework
2 includes object means for displaying alternate homophones in response
3 to the user input.
- 1 6. The apparatus of claim 1, wherein the input methods framework
2 includes object means for replacing displayed information with a user-
3 selected alternative.

-26-

- 1 7. The apparatus of claim 1, wherein the input methods framework
2 includes means for user confirmation of the converted user input.
- 1 8. The apparatus of claim 1, wherein the input methods framework
2 includes at least one object for providing dictionary services in response
3 to said user input.
- 1 9. The apparatus of claim 8, wherein the at least one object means for
2 providing dictionary services includes properties used in providing said
3 dictionary services.
- 1 10. The apparatus of claim 8, wherein the at least one object means for
2 providing dictionary services includes means for reverse mapping.
- 1 11. The apparatus of claim 8, wherein the at least one object for providing
2 dictionary services includes means for chaining dictionaries.
- 1 12. The apparatus of claim 1, including a system level which includes the
2 input methods framework.
- 1 13. The apparatus of claim 1, wherein the input methods framework
2 includes at least one object for grouping commands.
- 1 14. The apparatus of claim 1, wherein the input methods framework
2 includes at least one object which is an interactive modifier.
- 1 15. The apparatus of claim 1, wherein the input methods framework
2 includes at least one object for chaining modifiers.
- 1 16. The apparatus of claim 1, wherein the input methods framework
2 includes at least one object for event mapping.

-27-

- 1 17. The apparatus of claim 1, wherein the input methods framework
2 includes at least one object for defining the appearance of an active area.
- 1 18. The apparatus of claim 1, wherein the input methods framework
2 includes at least one object for coordinating multiple active areas.
- 1 19. The apparatus of claim 1, wherein the input methods framework
2 includes at least one object for coordinating multiple input methods.
- 1 20. The apparatus of claim 1, wherein the input methods framework
2 includes a conversion object.
- 1 21. A method for assisting in the input of information comprising:
2 (a) receiving user input into an input methods framework;
3 (b) in response to said user input, converting said user input with the input
4 methods framework;
5 (c) displaying the converted user input;
6 (d) repeating the above steps until no further converting is necessary.
- 1 22. The method of claim 1, including the step of processing a phrase in
2 response to said user input.
- 1 23. The method of claim 22, including the step of processing a user-selected
2 phrase.
- 1 24. The method of claim 22, including the step of changing a phrase
2 boundary of the phrase.
- 1 25. The method of claim 21, including the step of displaying alternate
2 homophones in response to the user input.
- 1 26. The method of claim 21, including the step of replacing displayed
2 information with a user-selected alternative.

-28-

- 1 27. The method of claim 21, including the step of user confirmation of the
2 converted user input.
- 1 28. The method of claim 21, including the step of providing dictionary
2 services in response to said user input.
- 1 29. The method of claim 28, including the step of providing properties used
2 in providing said dictionary services.
- 1 30. The method of claim 28, including the step of reverse mapping.
- 1 31. The method of claim 28, including the step of chaining dictionaries.
- 1 32. The method of claim 21, including the step of defining system levels
2 associated with a plurality of frameworks.
- 1 33. The method of claim 21, including the step of grouping commands.
- 1 34. The method of claim 21, including the step of requiring user interaction
2 with respect to a modification.
- 1 35. The method of claim 21, including the step of chaining modifiers.
- 1 36. The method of claim 21, including the step of event mapping.
- 1 37. The method of claim 21, including the step of defining the appearance of
2 an active area.
- 1 38. The method of claim 21, including the step of coordinating multiple
2 active areas.

-29-

- 1 39. The method of claim 21, including the step of coordinating multiple
- 2 input methods.

1 / 12

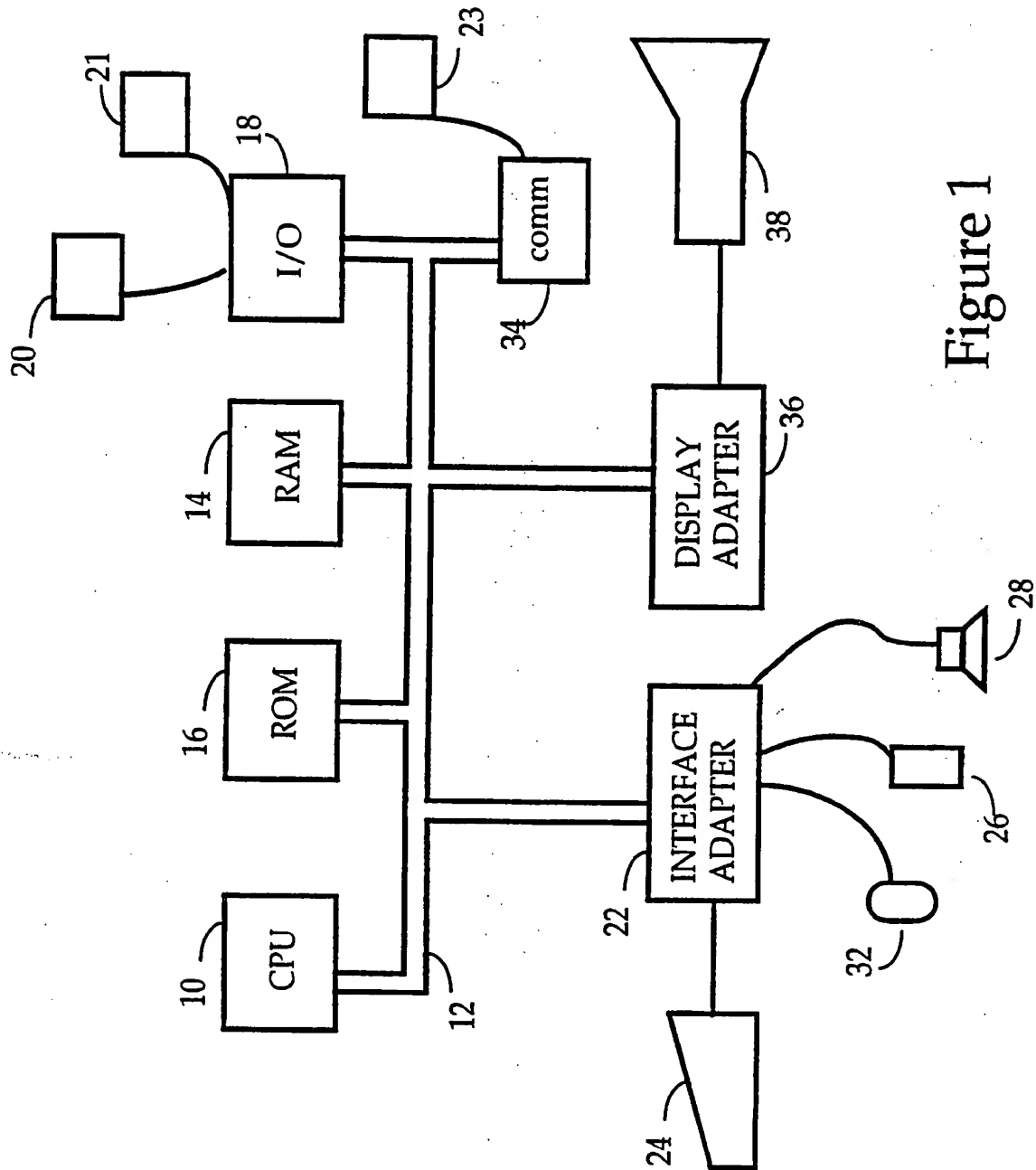


Figure 1

2/12

FAKE LOGOGRAPH

ENGLISH

CHINESE LOGOGRAPH

1

a/an

一

人 屋

neighbor
(man/house)鄰居
(side/dwell)

讀

read

讀

她

she/her

她

地址

address

地址
(ground/location)

Figure 2

3/12










TYPE-IN	SCREEN DISPLAY	READING
u		A
un		An
una	a	A· na
unay		A· neigh
unayb	 b	A· neigh-b
unaybu	 bu	A· neigh-bo
unaybur	 𐄂	A· neighbor
unayburr	 𐄂 r	A· neighbor-r
unayburre	 𐄂 re	A· neighbor-re
unayburred	 𐄂 𐄂	A· neighbor· red
...		...
unayburredhur	 𐄂 𐄂 𐄂	A· neighbor· red· her
...		...
unayburredhurudres	 𐄂 𐄂 𐄂 1 𐄂	A· neighbor· read· her· a · dress

Figure 3

4/12

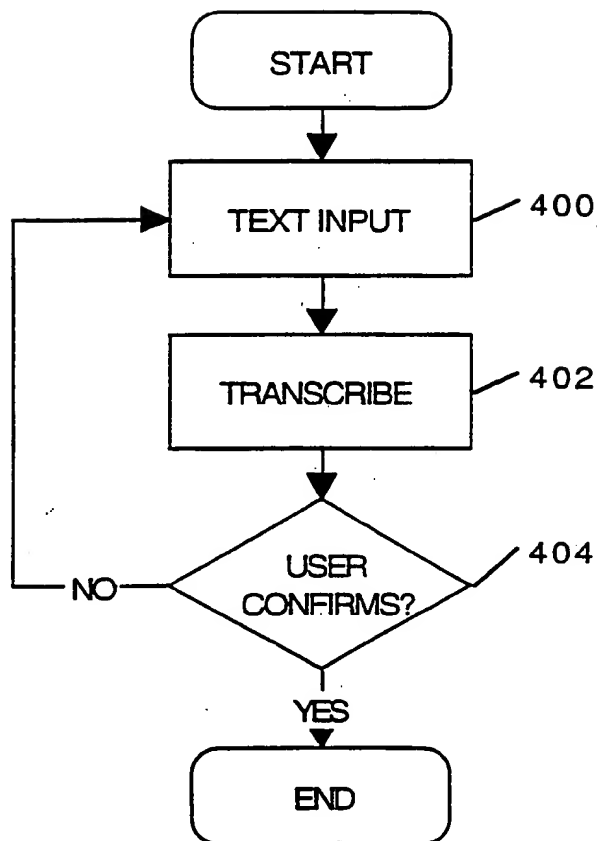


Figure 4

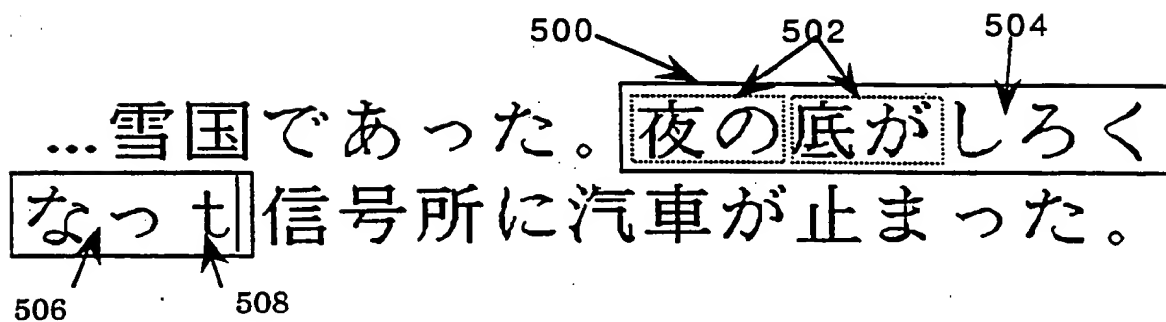


Figure 5

5/12

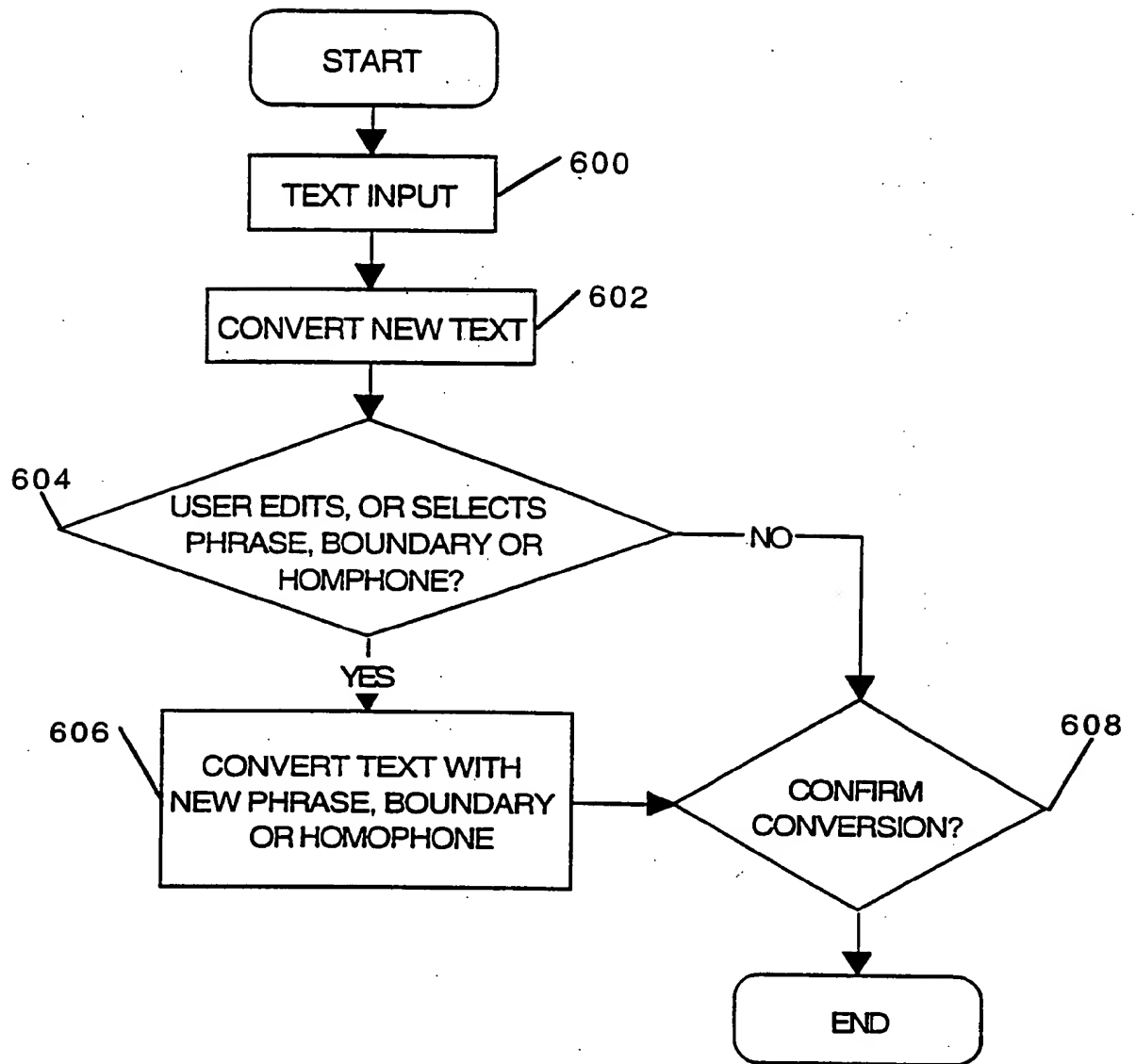


Figure 6

6/12

はるにはながさき → 春には長崎

(a)

はる → 1. 春 2. 晴 3. 張る 4. 貼る

(b)

zhang3 長

chang2 長

(c)

Figure 7

7/12

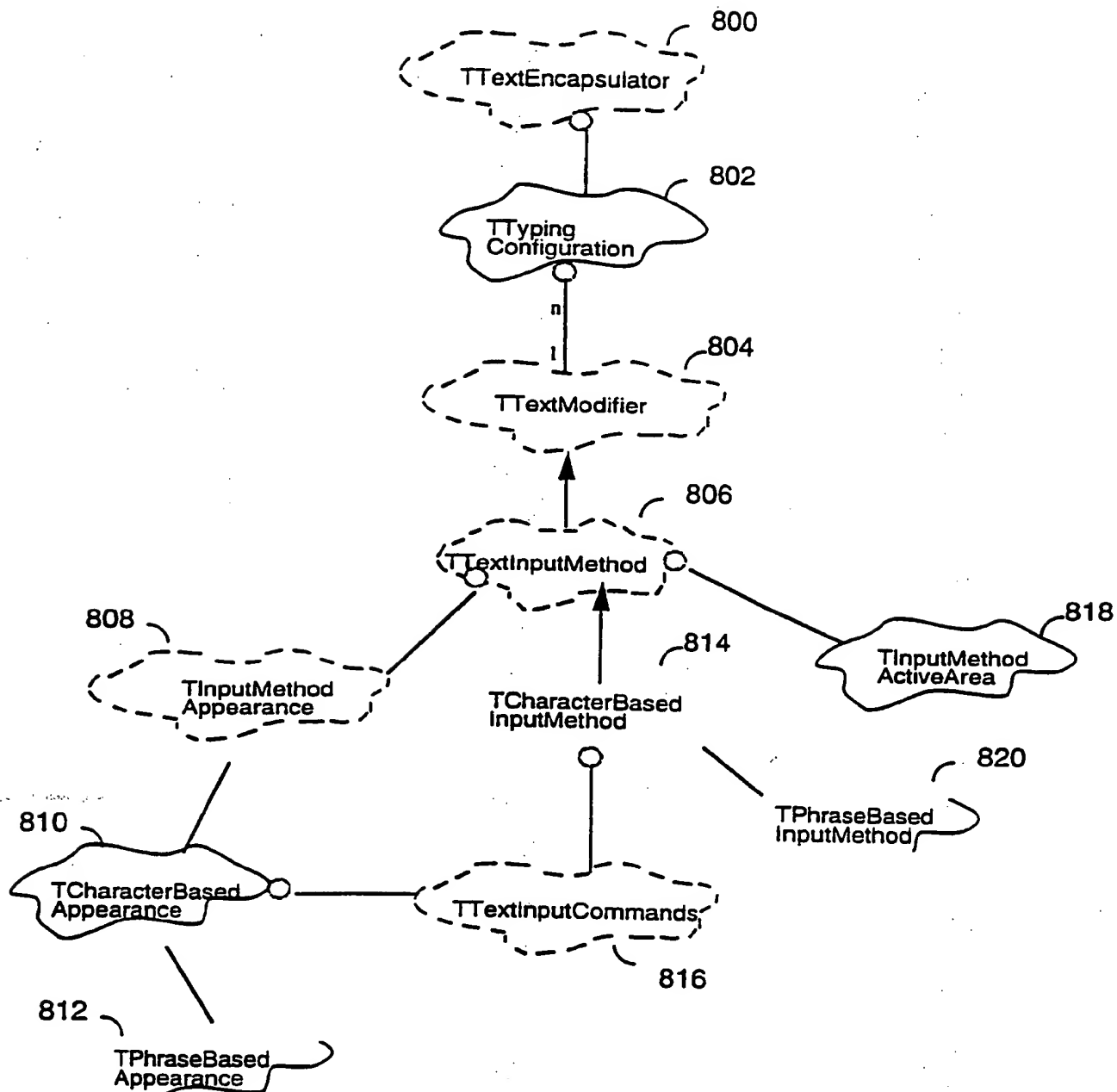


Figure 8

8/12

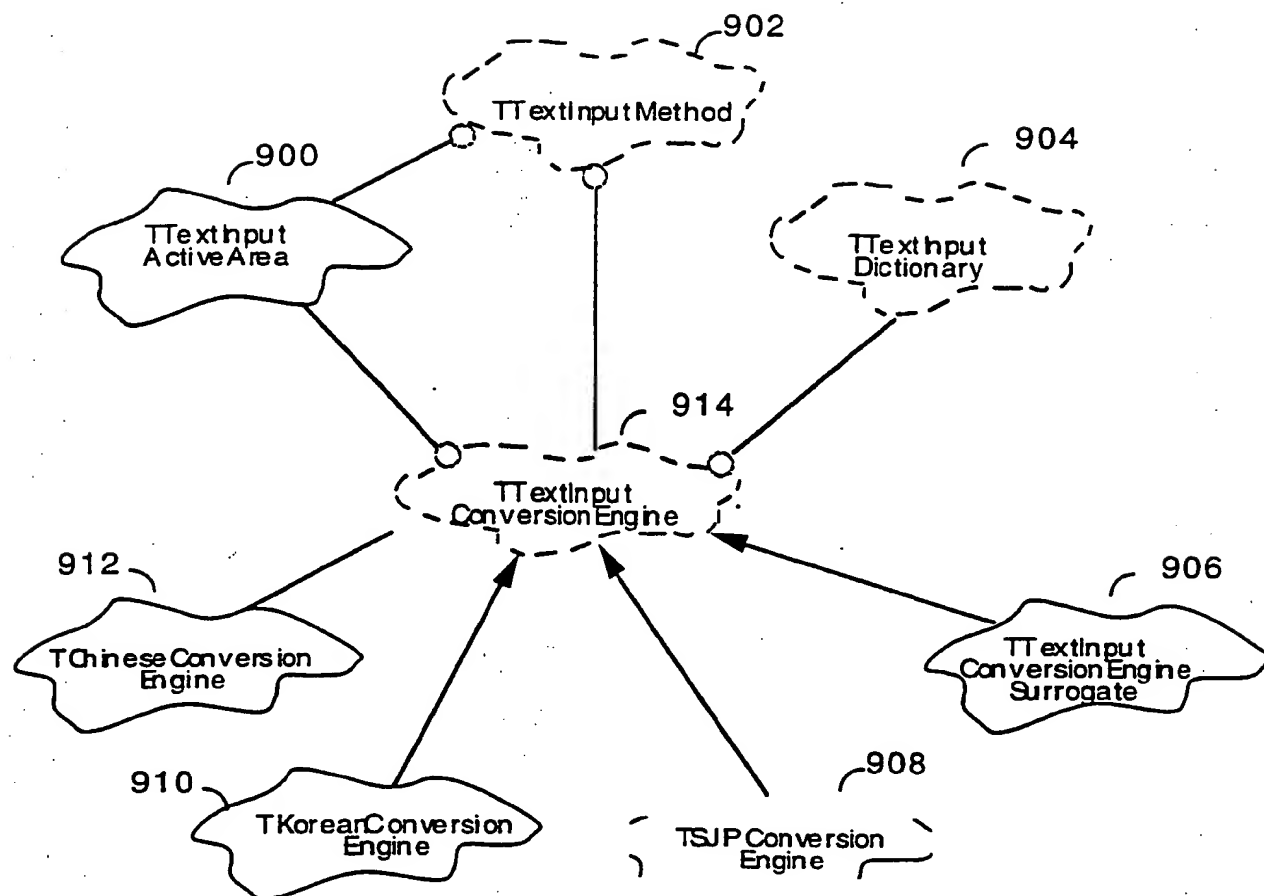


Figure 9

9/12

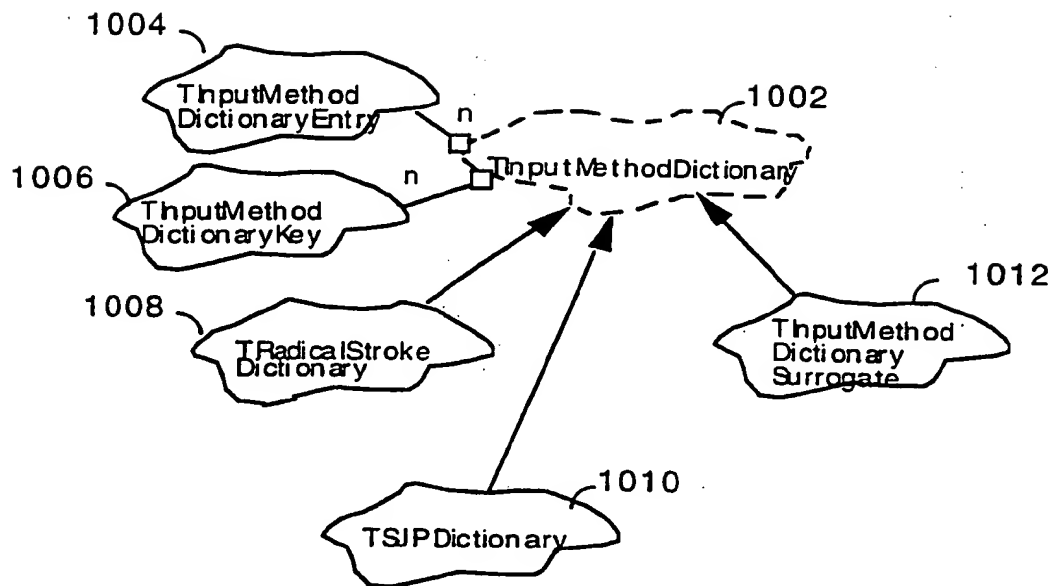


Figure 10

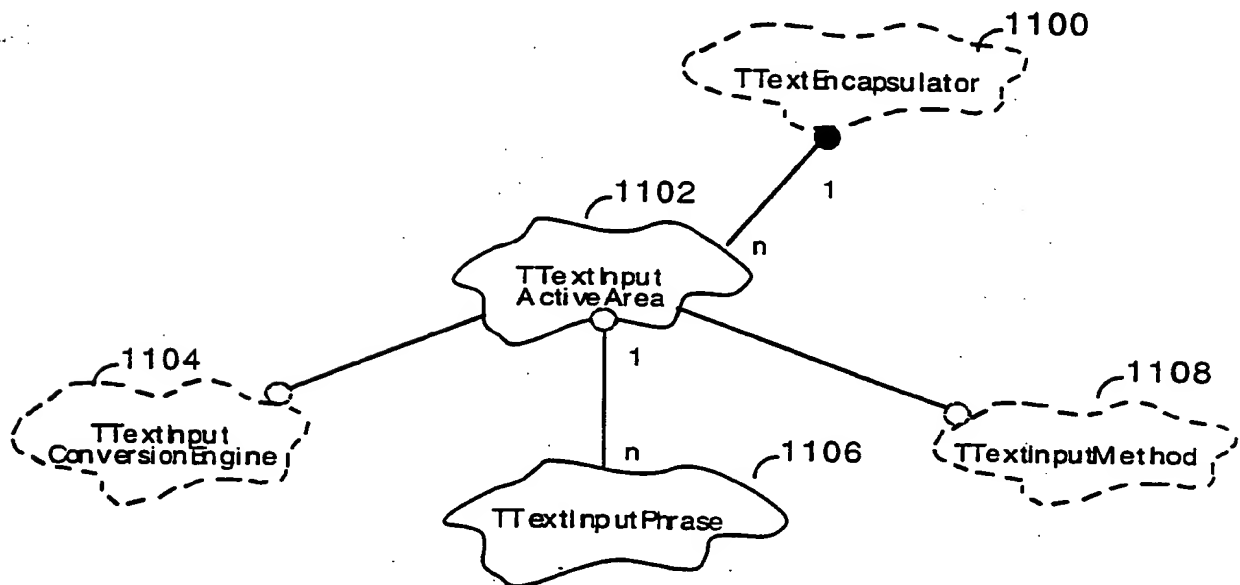


Figure 11

10/12

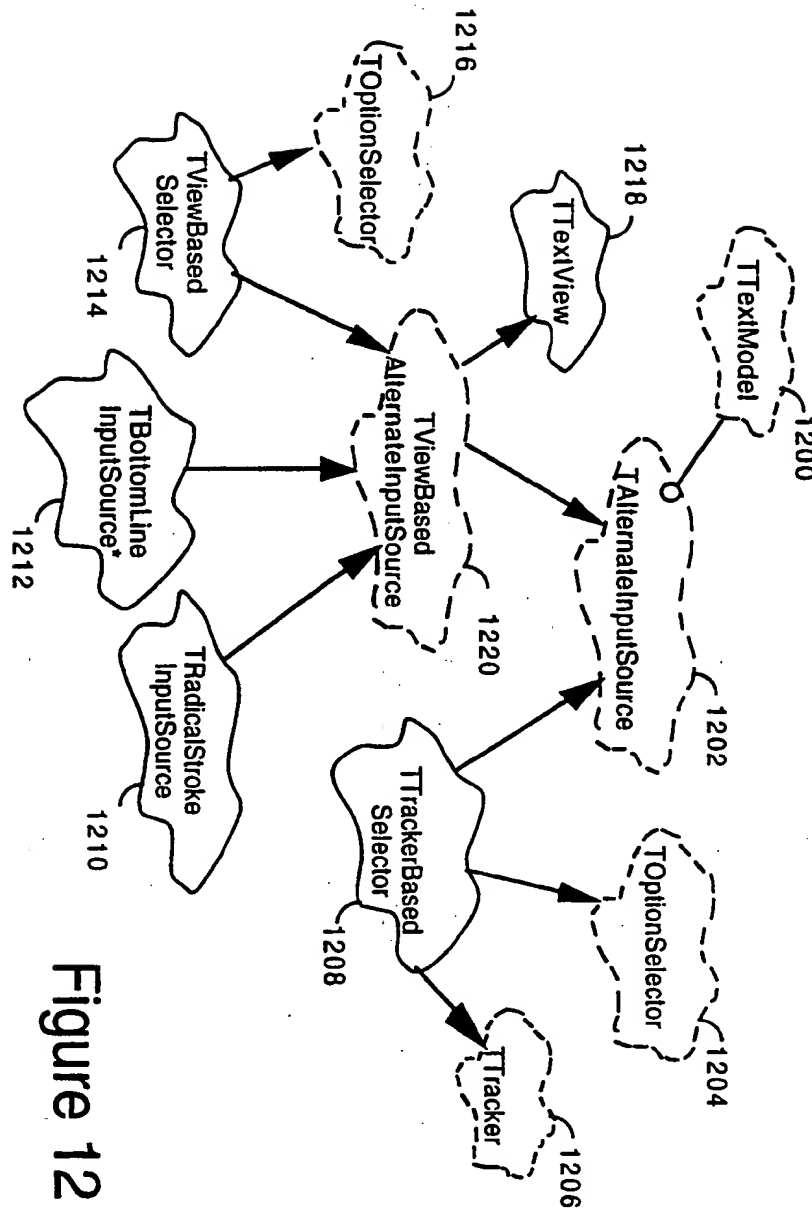


Figure 12

11/12

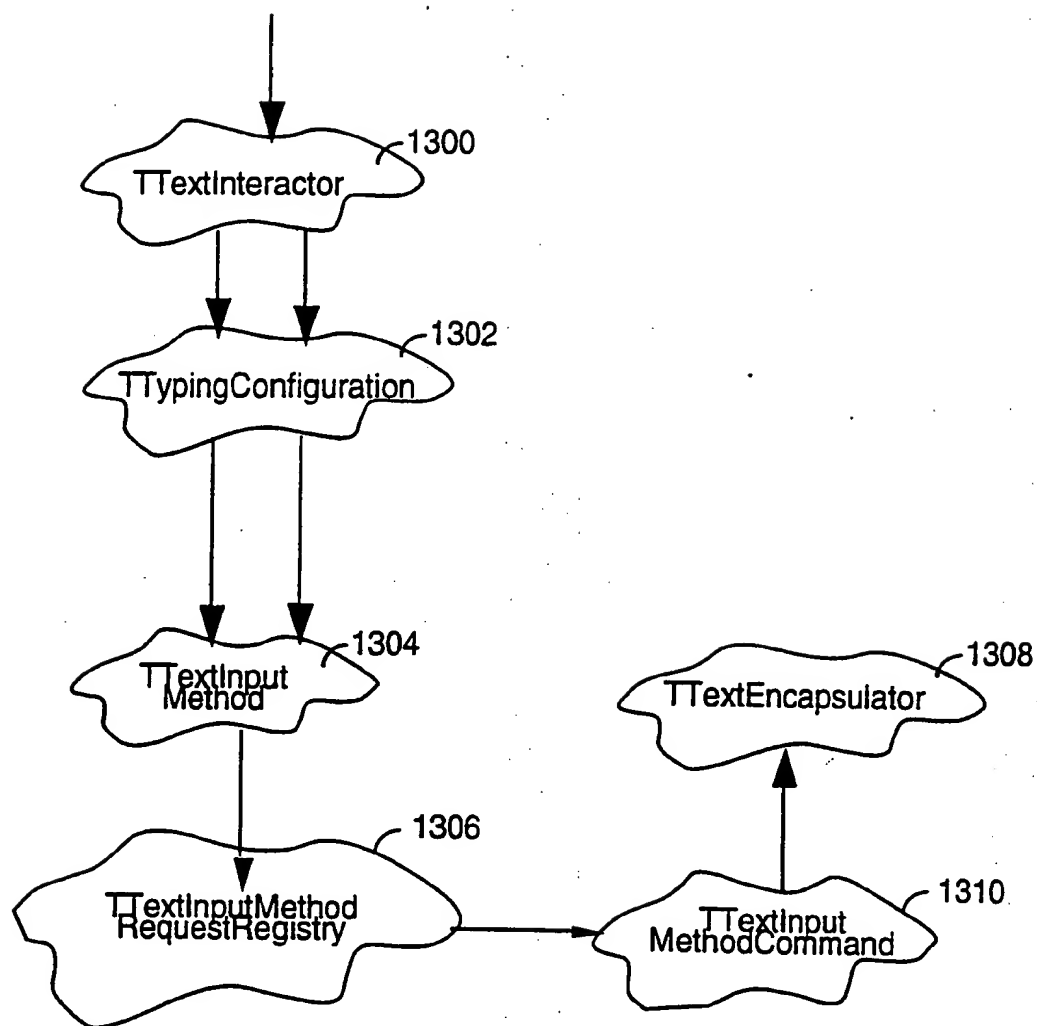


Figure 13

12/12

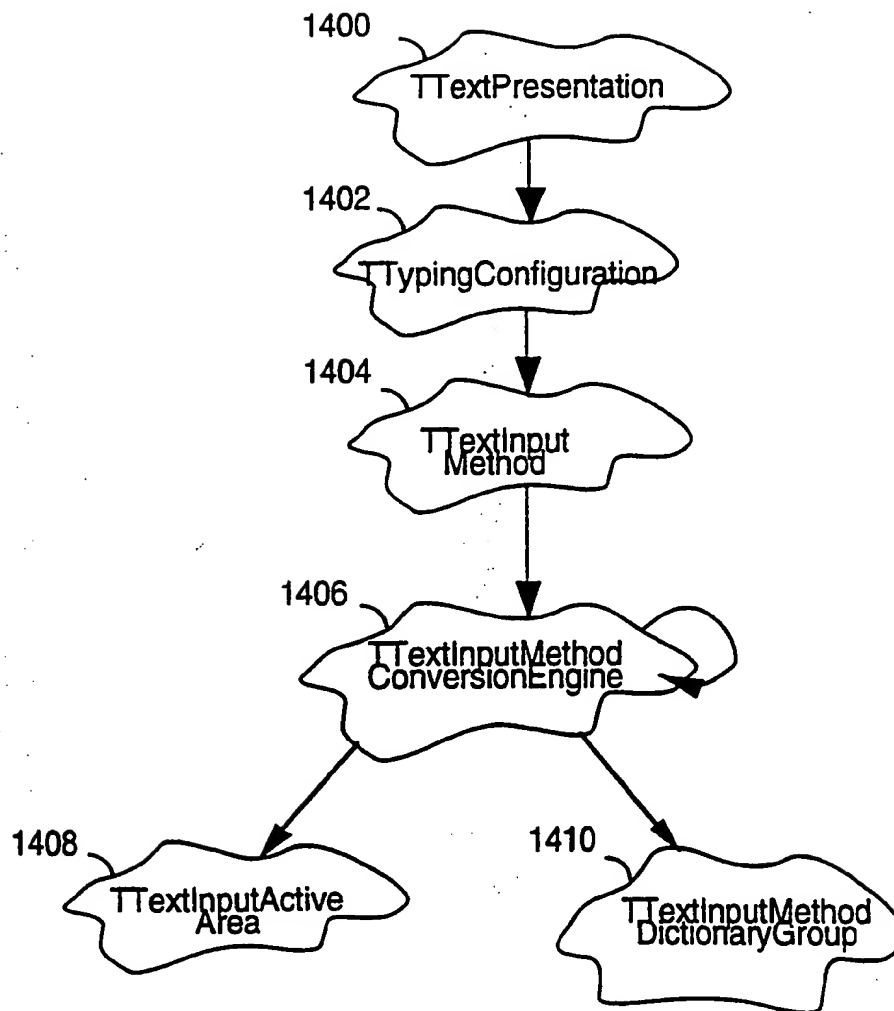


Figure 14

Intern I Application No
PCT/US 94/05586

According to International Patent Classification (IPC) or to both national classification and IPC

Minimum documentation searched (classification system followed by classification symbols)

IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	<p>PROCEEDINGS OF THE FOURTH ANNUAL SYMPOSIUM ON USER INTERFACE SOFTWARE AND TECHNOLOGY (UIST'91), November 1991, HILTON HEAD, USA pages 185 - 193, XP000315079</p> <p>M. MORISAKI ET AL 'XJp System: An Internationalized Language Interface for the X Window System'</p> <p>see page 187 - page 193</p> <p>----</p>	1,21
A	<p>ACM TRANSACTIONS ON INFORMATION SYSTEMS, vol.10, no.4, October 1992, NEW YORK, US pages 438 - 451, XP000362343</p> <p>Y. KATAOKA ET AL 'A Model for Input and Output of Multilingual Text in a Windowing Environment'</p> <p>see the whole document</p> <p>----</p> <p style="text-align: center;">-/--</p>	1-39

X Further documents are listed in the continuation of box C.

Y Patent family members are listed in annex.

* Special categories of cited documents :

- * **A** document defining the general state of the art which is not considered to be of particular relevance
- * **E** earlier document but published on or after the international filing date
- * **L** document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- * **O** document referring to an oral disclosure, use, exhibition or other means
- * **P** document published prior to the international filing date but later than the priority date claimed

- "T"** later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
- "X"** document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
- "Y"** document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
- "&"** document member of the same patent family

Date of the actual completion of the international search

22 November 1994

Date of mailing of the international search report

01.12.94

Name and mailing address of the ISA
European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,
Fax: (+31-70) 340-3016

Authorized officer

Pottiez, M

INTERNATIONAL SEARCH REPORT

Inten I Application No
PCT/US 94/05586

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	IBM TECHNICAL DISCLOSURE BULLETIN, vol.36, no.11, November 1993, ARMONK, US pages 629 - 638, XP000424975 'National Language Support Enablement for Culture-Specific Operations' see the whole document ---	1-39
A	EP,A,0 398 646 (IBM) 22 November 1990 see the whole document -----	1,21

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

information on patent family members

Inter.

Application No

PCT/US 94/05586

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP-A-0398646	22-11-90	CA-A- 2016225	15-11-90
		JP-A- 3006651	14-01-91
		KR-B- 9402343	23-03-94
